Karlsruher Institut für Technologie

Programmieren Tutorium 11 – Interfaces und Generics

Institut für Zeritfizierbare und Vertrauenswürdige Informatiksysteme (ZVI)

Inhalt

Interfaces

Karlsruher Institut für Technologie

- Enthalten Konstanten und Methoden
- Alle Methoden Abstrakt
- Erben nur von Interfaces

Interfaces

Karlsruher Institut für Technologie

- Enthalten Konstanten und Methoden
- Alle Methoden Abstrakt
- Erben nur von Interfaces
- Mehrfachvererbung sicher

Interfaces

```
public interface Interface extends Cloneable {
    public static final int CONSTANT = 0;
    public boolean foo();
}

public class Bar implements Interface {
    @Override
    public boolean foo() {
        return false;
    }
}
```

Generics

Karlsruher Institut für Technologie

- Verhindert Codeduplikate
- Typsicher (meistens)
- Prüfung der Typen durch Compiler (oder auch nicht!)

Generics

- Verhindert Codeduplikate
- Typsicher (meistens)
- Prüfung der Typen durch Compiler (oder auch nicht!)

`http://goo.gl/ZumuL`, Section 4.4

Vergessene Typerweiterung

Karlsruher Institut für Technologie

```
LinkedList is = new LinkedList<Integer>();
```

Unchecked-Warnung abgeschaltet

Karlsruher Institut für Technologie

```
@SuppressWarnings("unchecked")
public LinkedList foo() {
    LinkedList<Integer> is = new LinkedList();
    // Do something
    return is;
}

@SuppressWarnings("unchecked")
public void bar() {
    LinkedList<Float> = foo();
}
```

Aufgabe

Karlsruher Institut für Technologie

Functional Programming in Java!

Aufgabe

Karlsruher Institut für Technologie

Functional Programming in Java!

Implementieren Sie ein Interface `FancyList`, das sich an das Interface `java.util.List` hält und ausserdem die Funktionen `foldr`, `foldl`, `map` und `filter` definiert. Implementieren Sie eine FancyList unter Verwendung von Vererbung. Schreiben Sie dazu Interfaces `Function1` und `Function2`, die eine Funktion mit einem bzw. zwei Parametern darstellen.

Definitionen

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f x [] = x
foldr f x (y:ys) = f y (foldr f x ys)

foldl :: (b -> a -> b) -> b -> [a] -> b
foldl f x [] = x
foldl f x (y:ys) = f (foldl f x ys) y

map :: (a -> b) -> [a] -> [b]
map f xs = foldr transform xs []
    where transform elem buf = (f elem) : buf

filter :: (a -> Bool) -> [a] -> [a]
filter p xs = foldr transform xs []
    where transform elem buf | p elem = elem : buf
                             | otherwise = buf
```

Ende

«Brains aside, I wonder how many poorly-written xkcd.com-parsing scripts will break on this title (or \;;";\’’{<<;[’ this mouseover text.";»